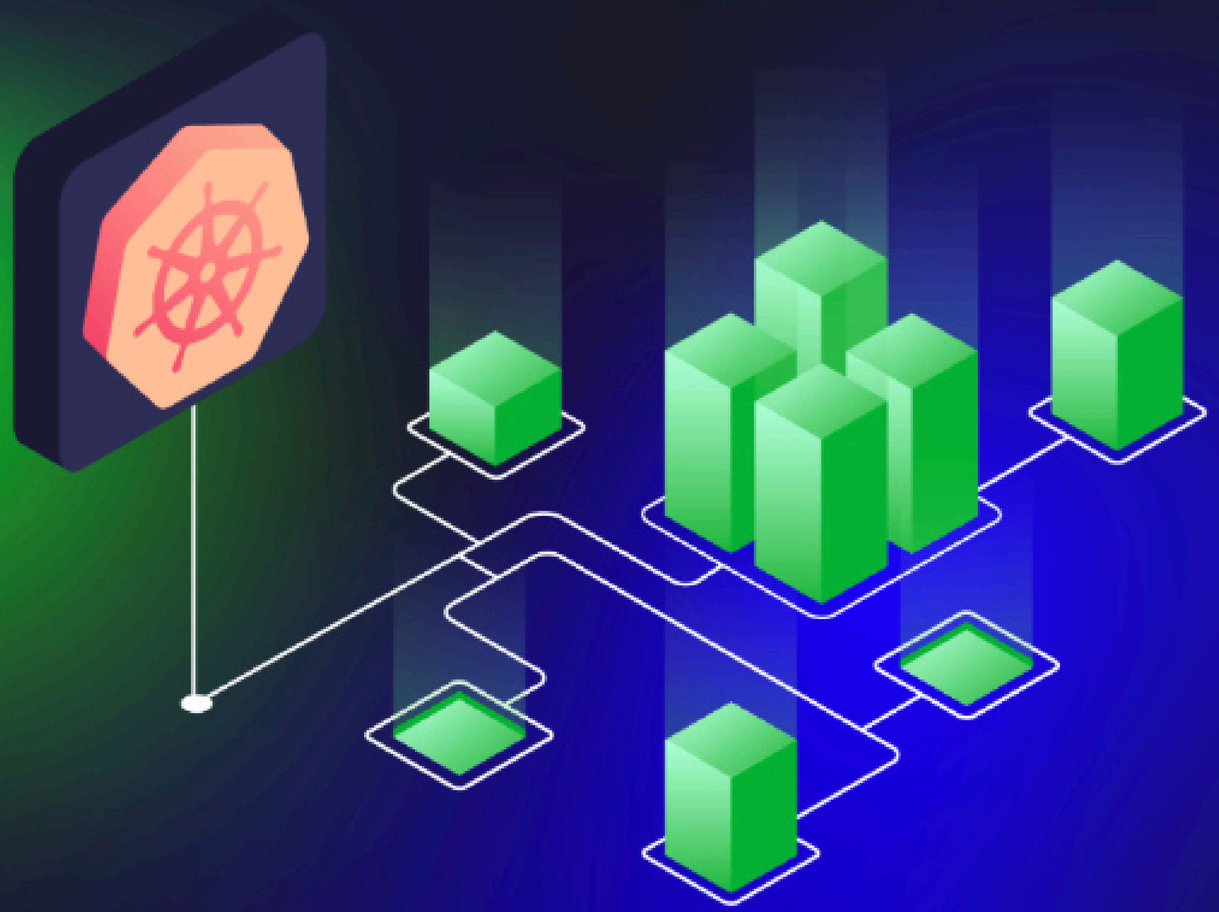
CHECKLIST



Azure Kubernetes Service (AKS) Checklist

Practical tips & best practices

Intro

Kubernetes has become the de facto platform for running modern applications, with production adoption reaching 82% and continuing to grow as AI and data workloads increasingly rely on containerised architectures.

As organisations scale these workloads, Azure Kubernetes Service (AKS) is emerging as a primary platform for running Kubernetes in the cloud, providing a managed environment for deploying, operating, and scaling containerised applications.

This checklist helps you prepare AKS clusters for production. It brings together key best practices across operations, security, reliability, performance, cost, and governance to assess whether your platform is ready to support production workloads.

Note

This checklist is intended to be a comprehensive reference rather than a strict standard. It covers the key considerations typically required to operate Kubernetes reliably in production and helps identify gaps that may affect stability, security, or scalability.

Not all recommendations in this checklist will apply in every scenario, and some may need to be adapted based on your specific context. Use this checklist to guide decisions, identify gaps, and validate your AKS platform design.

Table of Content

1. Operational Excellence

Operate AKS using automation, monitoring, and well-defined incident management processes. Review operational metrics regularly and adjust configurations as the platform and workloads evolve.

Implement cluster monitoring with Azure Monitor Container Insights
Enable Container Insights to collect metrics, logs, and performance data from AKS clusters, nodes, and containers, integrated with Log Analytics for centralised monitoring and alerting. This provides real-time visibility into utilisation, performance, and cluster health, enabling proactive issue detection.

Establish structured logging with centralised log aggregation
Forward container logs to Azure Log Analytics or Event Hubs and use structured formats such as JSON to enable efficient querying across services. This ensures complete audit trails and accelerates troubleshooting during incidents.

Deploy infrastructure as code (IaC)
Manage AKS infrastructure declaratively using tools like Bicep or Terraform, with all definitions stored in Git for version control and review. This removes configuration drift and enables consistent environment recreation.

Implement GitOps for app deployment and configuration management
Use tools such as Flux or ArgoCD to manage Kubernetes manifests and configurations from Git repositories. This maintains a single source of truth, enables automated reconciliation, and reduces deployment errors and drift.

Configure auto-scaling at multiple levels
Enable Cluster Autoscaler and Horizontal Pod Autoscaler based on CPU, memory, or custom metrics, and use Vertical Pod Autoscaler where appropriate. This aligns resource usage with demand while maintaining performance and avoiding over-provisioning.

Establish proactive alerting with Azure Monitor and Prometheus
Create alerts for node health, pod failures, resource exhaustion, and API server latency, and use Prometheus with Grafana for custom metrics and visualisation. This enables early issue detection and reduces time to identify problems.

Implement a comprehensive tagging strategy for resource management
Apply consistent tags across AKS resources and enforce standards with Azure Policy to support structured reporting. This enables cost allocation, ownership tracking, and governance across environments.

Maintain cluster upgrade cadence and node image updates
Regularly upgrade Kubernetes versions and node images using maintenance windows and surge upgrades to minimise disruption. This ensures supported configurations, applies security patches, and limits technical debt.

Deploy distributed tracing for microservices observability
Use Application Insights with OpenTelemetry to trace requests across services. This improves visibility into dependencies, helping identify bottlenecks and reduce resolution time.

Establish incident management processes with runbooks
Define runbooks for common failure scenarios and establish escalation paths and on-call procedures integrated with alerting systems. This ensures consistent handling of incidents and reduces recovery time.

Implement capacity planning with regular resource analysis
Review usage trends using Azure Monitor and Azure Advisor, analysing historical patterns and growth. This prevents resource shortages and avoids unnecessary over-provisioning.

Maintain service dependency mapping and documentation
Document service dependencies, integrations, and data flows, and keep architecture diagrams and API contracts up to date in a central repository. This improves troubleshooting, impact analysis, and recovery planning.

Automate backup and disaster recovery procedures
Implement automated backups using Azure Backup for AKS and regularly test restore processes alongside cluster reconstruction. This ensures reliable recovery and reduces impact during failures.

Enforce configuration drift detection and remediation
Use Azure Policy to detect drift from the desired state and trigger remediation or alerts. This maintains consistency, enforces standards, and prevents unauthorised changes.

Establish performance baselines and SLI/SLO tracking
Define SLIs and SLOs and track them using Azure Monitor or Grafana, establishing baselines during normal operation. This enables objective performance measurement and early detection of deviations.

2. Security

Secure AKS by protecting identities, workloads, and network boundaries. Implement controls to prevent unauthorised access, detect threats, and reduce the attack surface across the platform.

Enable Microsoft Entra ID (Azure AD) integration for RBAC
Integrate AKS with Microsoft Entra ID to enable Kubernetes RBAC using Entra groups and enterprise identity policies. This centralises authentication, enforces conditional access, and provides full auditability.

Implement workload identity using Microsoft Entra Workload ID
Use Microsoft Entra Workload ID to assign managed identities to pods, avoiding service principals and stored credentials. This provides secure, scalable authentication using federated identities and is the recommended approach for AKS.

Enable Azure Policy for Kubernetes and enforce pod security standards
Integrate AKS with Azure AD to enable Kubernetes RBAC using Azure AD groups and enterprise identity policies. This centralises authentication, enforces conditional access, and provides full auditability.

Secure network traffic with Azure CNI and network policies
Use Azure CNI to assign VNet IPs to pods and integrate with network security groups and Azure Firewall, and apply network policies for micro-segmentation. This restricts communication to authorised paths and reduces lateral movement.

Enable private cluster configuration with private endpoint
Configure AKS as a private cluster with API server access limited to private endpoints within the virtual network. This removes public exposure and restricts control plane access.

Implement image scanning and trusted registry policies
Scan images using Microsoft Defender for Containers and restrict deployments to trusted registries such as Azure Container Registry (ACR). This prevents vulnerable or unauthorised images from being deployed.

Enable Microsoft Defender for Containers for runtime protection
Use Defender for Containers for runtime threat detection, vulnerability assessment, and security recommendations. This enables continuous monitoring and detection of suspicious activity.

Encrypt secrets at rest with Azure Key Vault integration
Use the Key Vault CSI Driver to mount secrets directly into pods without storing them in etcd. This centralises secret management and enforces secure storage and access controls.

Implement just-in-time (JIT) cluster access with Azure Bastion
Disable direct SSH and enable just-in-time access through Azure Bastion or Privileged Identity Management. This ensures time-bound, controlled, and auditable administrative access.

Enable audit logging and integrate with Azure Monitor
Send Kubernetes audit logs to Azure Monitor Log Analytics for centralised analysis and retention. This provides visibility into cluster activity and supports investigations and compliance.

Implement pod security context and admission controllers
Define security contexts to enforce least privilege and use admission controllers such as Azure Policy to validate configurations. This prevents privilege escalation and limits blast radius.

Secure node access with Azure Disk Encryption and OS hardening
Enable disk encryption for node OS and data disks and apply OS hardening and regular patching. This reduces node-level vulnerabilities and protects stored data.

Implement egress traffic control with Azure Firewall
Route outbound traffic through Azure Firewall and apply filtering with threat intelligence. This enables inspection and prevents unauthorised external communication.

Establish certificate management with cert-manager & Key Vault
Use cert-manager with Azure Key Vault to automate TLS certificate lifecycle management. This ensures certificates are rotated, securely stored, and consistently applied.

Enable service mesh for mTLS with Azure Service Mesh or Istio
Use a service mesh to enforce mutual TLS between services and apply fine-grained traffic policies. This secures service-to-service communication without application changes.

Use managed identities instead of service principals
Use managed identities for clusters and workloads to avoid manual credential management. This reduces credential exposure and aligns with Azure-native practices.

Disable local accounts in AKS
Disable local Kubernetes accounts to enforce Entra ID-based authentication. This ensures all access is centrally controlled and auditable.

Restrict access to admin kubeconfig
Limit use of admin credentials using Azure RBAC to control access to cluster configuration. This prevents unrestricted access and enforces least privilege.

Use AKS–ACR integration without credentials
Enable managed identity-based integration between AKS and Azure Container Registry. This removes the need to store registry credentials in the cluster.

Use kubelogin for non-interactive access
Use kubelogin for secure authentication in automation scenarios such as CI/CD pipelines. This avoids long-lived credentials and improves access security.

3. Reliability

Ensure AKS configurations support high availability and fault tolerance across clusters and workloads. Validate failover behaviour and disaster recovery procedures under realistic conditions to confirm the platform remains operational during failures.

Implement multi-zone node pools for high availability
Deploy AKS node pools across multiple availability zones to protect against datacentre failures and maintain service availability during a zone outage. This supports zone-redundant architecture for production workloads requiring 99.95% or higher SLA guarantees.

Configure pod disruption budgets (PDBs)
Define Pod Disruption Budgets to maintain a minimum number of replicas during voluntary disruptions such as maintenance or upgrades. This prevents all instances of a critical workload from being terminated at the same time and preserves availability during cluster operations.

Enable AKS cluster autoscaler
Configure the cluster autoscaler to adjust node capacity automatically based on resource demand. This prevents pod scheduling failures during traffic spikes while reducing wasted capacity during low-demand periods.

Implement horizontal pod autoscaling (HPA)
Configure HPA to scale application replicas based on CPU, memory, or custom metrics. Combined with cluster autoscaling, this maintains application performance under changing demand at both pod and node level.

Use Azure Backup for AKS for disaster recovery
Implement backups of cluster state, persistent volumes, and application configuration using Azure Backup for AKS, and test recovery procedures regularly. This supports recovery from data corruption, accidental deletion, or major failures while helping meet RPO & RTO requirements.

Configure resource requests and limits
Define CPU and memory requests and limits for all pods to guarantee minimum resources and prevent excessive consumption by individual workloads. This improves cluster stability and reduces the risk of resource contention and cascading failures.

Implement health probes (liveness and readiness)
Configure liveness probes to restart unhealthy containers and readiness probes to ensure only healthy pods receive traffic. This enables automatic recovery from failures and prevents traffic being routed to instances that are not ready to serve requests.

Enable AKS uptime SLA for production workloads
Enable the AKS Uptime SLA for production clusters to obtain a financially backed availability guarantee. This provides 99.95% availability, or 99.99% with availability zones, through redundant control plane components and Azure’s availability commitments.

Implement node pool surge upgrades
Configure max surge settings to control additional node capacity during upgrades and balance upgrade speed with available resources. This allows upgrades to complete without exhausting capacity or disrupting running workloads.

Use Azure Container Registry with geo-replication
Store container images in Azure Container Registry with geo-replication enabled to improve image availability across regions. This reduces dependency on a single region and helps prevent deployment failures during regional outages.

Monitor and configure alerts
Enable Container Insights to collect metrics, logs, and performance data, and configure alerts for node pressure, pod failures, and resource exhaustion. This supports early detection of reliability issues and faster response before application availability is affected.

Use network policies for resilient microservices communication
Implement network policies to control pod-to-pod communication and implement retry logic through service meshes to handle transient failures. This limits failure propagation between services and improves resilience during dependency issues.

Plan for service quotas and request increases proactively
Monitor quotas such as nodes per cluster, pods per node, and IP address limits, and request increases before limits are reached. This avoids situations where scaling or recovery is blocked at critical moments.

Implement StatefulSet anti-affinity rules
Configure pod anti-affinity rules for StatefulSets to distribute replicas across nodes and availability zones. This prevents a single node or zone failure from affecting all replicas of a stateful workload.

Regularly test disaster recovery procedures
Conduct DR tests covering cluster restoration, failover to secondary regions, and data recovery from backups. This validates recovery procedures, exposes gaps in tooling and process, and prepares for real incidents.

Plan for multiregion deployment
Design workloads to run across multiple Azure regions where recovery requirements demand it, using paired regions and suitable replication strategies. This protects against regional outages and supports full disaster recovery.

Use global traffic management for failover
Use Azure Traffic Manager or Azure Front Door to route traffic across regions and direct users to healthy clusters automatically. This improves regional failover and maintains service availability during major outages.

Avoid storing application state in the cluster
Use external Azure PaaS services such as Azure SQL, Cosmos DB, or Azure Storage for stateful data where possible. This improves resilience, supports replication, and simplifies recovery compared with in-cluster state.

Use ephemeral OS disks for AKS nodes where appropriate
Use ephemeral OS disks to reduce OS disk latency and improve node reimaging, scale operations, and upgrade speed. This shortens recovery and maintenance operations on AKS agent nodes.

Align storage strategy with availability zones
Use zone-aware storage design when combining Azure Disks with availability zones, including appropriate use of LRS, ZRS. This ensures volumes are provisioned in the correct zone and avoids storage becoming a point of failure.

4. Performance Efficiency

Optimise performance by monitoring workload behaviour, right-sizing resources, and reducing latency across the platform. Continuously review and adjust configurations to maintain throughput and avoid bottlenecks.

Implement Horizontal Pod Autoscaler (HPA) with custom metrics
Configure HPA to scale pods automatically based on CPU, memory, and custom metrics from Azure Monitor. This aligns scaling decisions with actual application behaviour and avoids over- or under-scaling based on incomplete signals.

Enable cluster autoscaler for node-level scaling
Use the cluster autoscaler to adjust node pool size based on pending pod requests and workload demand. This ensures sufficient capacity during peaks while avoiding idle infrastructure during low usage.

Use Azure CNI with IP address planning
Implement Azure CNI to assign VNet IP addresses directly to pods and enable efficient communication with Azure resources. This reduces latency and avoids network bottlenecks, while requiring careful subnet planning to prevent IP exhaustion.

Optimise node pool sizing with multiple node pools
Use separate node pools for different workload types such as CPU-intensive, memory-intensive, or GPU workloads. This ensures workloads run on appropriately sized infrastructure and avoids inefficient resource allocation.

Configure resource requests and limits appropriately
Define accurate CPU and memory requests and limits for all containers to support efficient scheduling and prevent resource contention. This allows the scheduler to place workloads optimally and improves overall cluster utilisation.

Use Azure Monitor Container Insights for performance monitoring
Enable Container Insights to collect and analyse performance metrics, logs, and health data from AKS clusters. This provides visibility into utilisation trends and helps identify performance bottlenecks early.

Use Premium or Ultra Disk storage for I/O-intensive workloads)
Select Premium SSD or Ultra Disk storage for workloads with high I/O requirements such as databases. This ensures consistent performance and avoids latency caused by storage bottlenecks.

Enable Azure Policy for resource governance
Use Azure Policy to enforce quotas, limit ranges, and prevent inefficient configurations such as missing limits or oversized requests. This maintains consistent performance standards and prevents workloads from degrading cluster efficiency.

Optimise container images and implement image caching
Reduce container image size using multi-stage builds and store images in ACR in the same region as the cluster. This reduces pull times, improves startup performance, and minimises scaling delays.

Configure pod disruption budgets for performance stability
Define Pod Disruption Budgets to maintain sufficient replicas during upgrades or scaling events. This prevents performance degradation by ensuring capacity is maintained during operational changes.

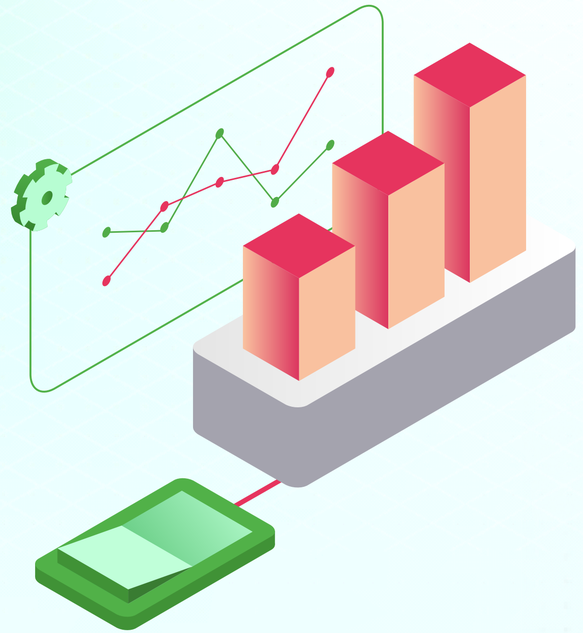
Use proximity placement groups for latency-sensitive workloads
Deploy latency-sensitive workloads using proximity placement groups to keep nodes physically close within Azure datacentres. This reduces network latency and improves performance for distributed applications.

Implement readiness and liveness probes correctly
Configure probes to ensure traffic is routed only to healthy pods and failed containers are restarted promptly. This prevents performance issues caused by unhealthy instances receiving traffic or unnecessary restarts.

Enable Kubernetes Event-Driven Autoscaling (KEDA)
Use KEDA to scale workloads based on event sources such as queues or custom signals beyond CPU and memory metrics. This enables efficient scaling, including scaling to zero during idle periods while maintaining responsiveness.

Optimise DNS performance with CoreDNS tuning
Configure CoreDNS caching, adjust replica counts, and use node-local DNS caching to reduce lookup latency. This prevents DNS from becoming a bottleneck in service-to-service communication.

Use Azure Spot Virtual Machines for fault-tolerant workloads
Deploy stateless or non-critical workloads on Spot VM node pools to reduce cost while maintaining acceptable performance. This allows cost optimisation without impacting critical workloads.



5. Cost Optimisation

Cost Optimisation within the framework helps organisations maximise the value they deliver while staying within financial constraints. It isn't just about reducing spend but about optimising cost in a way that supports business objectives and avoids waste. Without proper cost management, cloud environments can quickly lead to budget overruns, unexpected expenditures, and inefficiencies that are difficult to rectify

Right-size node pools based on workload requirements
Analyse CPU and memory utilisation using Azure Monitor to select appropriate VM sizes for node pools and adjust them based on actual demand. This avoids wasted capacity from oversized nodes while preventing performance issues caused by under-provisioning.

Implement cluster autoscaler for dynamic scaling
Enable the cluster autoscaler to adjust node count based on workload demand and pending pod requests. This reduces cost during low usage while ensuring sufficient capacity during traffic spikes.

Use Azure Spot Virtual Machines for fault-tolerant workloads
Deploy stateless or batch workloads on Spot VM node pools to take advantage of discounted compute capacity. This reduces cost significantly while maintaining acceptable performance for workloads that tolerate interruption.

Leverage Horizontal Pod Autoscaler (HPA) to optimise pod counts
Use HPA to scale pod replicas based on resource usage or custom metrics to match actual demand. This avoids running excess pods that consume resources and trigger unnecessary node scaling.

Set resource requests and limits accurately
Define realistic CPU and memory requests and limits based on observed usage patterns. This improves bin packing, reduces idle capacity, and prevents inefficient use of node resources.

Use Azure Reservations for predictable workloads
Apply Azure Reservations for workloads running consistently on the same VM SKUs and regions. This reduces long-term compute costs while maintaining flexibility through autoscaling for variable demand.

Enforce cost controls with Azure Policy
Use Azure Policy to restrict expensive VM sizes, enforce tagging, and apply quotas across AKS resources. This prevents uncontrolled spend and ensures deployments align with organisational cost standards.

Optimise storage costs with appropriate disk tiers
Use lower-cost storage options such as Standard HDD or SSD for non-critical workloads and reserve Premium tiers for high I/O requirements. This reduces storage spend without impacting performance where it matters.

Reduce data transfer costs by keeping traffic regional
Minimise cross-region and outbound traffic by placing dependent services within the same Azure region. This reduces network costs and avoids unnecessary latency between components.

Use lifecycle management policies for storage
Configure lifecycle policies to move data between Hot, Cool, and Archive tiers based on access patterns. This reduces storage costs automatically without manual intervention.

Dynamically provision storage volumes
Use Persistent Volume Claims with dynamic provisioning instead of pre-allocating storage. This ensures storage is created only when needed and avoids over-provisioning.

Schedule dev/test clusters outside business hours
Automate start and stop schedules for non-production clusters using Azure Automation or pipelines. This reduces cost by avoiding idle infrastructure during nights and weekends.

Optimise network costs with private connectivity
Deploy AKS clusters close to dependent services and use Private Link or service endpoints to keep traffic within Azure. This avoids egress charges and reduces unnecessary data transfer costs.

Identify and remove idle resources using Azure Advisor
Review Azure Advisor recommendations to detect underutilised resources such as node pools, disks, and load balancers. This removes unused components that continue to incur cost without delivering value.

Use multiple node pools with appropriate VM sizes
Use different node pools and VM SKUs to match workload requirements using selectors or affinity rules. This prevents small workloads from running on oversized nodes and improves cost efficiency.

Use Azure Cost Management for cost visibility
Enable cost analysis, budgets, and tagging to track AKS spend across clusters, namespaces, and teams. This provides visibility into usage patterns and highlights optimisation opportunities early.

Implement FinOps practices with showback and chargeback
Map namespaces and tags to teams or projects and establish showback or chargeback models. This increases cost accountability and encourages teams to optimise their resource usage.

6. Compliance & Governance

Ensure Azure Kubernetes Service aligns with organisational policies and regulatory requirements. Enforce standards, maintain auditability, and establish control over how clusters are configured and operated.

Implement Azure Policy for AKS
Use Azure Policy to enforce organisational standards and assess compliance across clusters using built-in and custom definitions. This ensures consistent configurations and enables continuous compliance monitoring and remediation.

Standardise cluster configuration across environments
Define baseline configurations for networking, security, scaling, and monitoring and apply them consistently across environments. This reduces variability and simplifies operations, audits, and troubleshooting.

Maintain centralised audit logging and retention policies
Ensure all control plane, audit, and operational logs are collected centrally and retained according to organisational requirements. This supports investigations, compliance reporting, and traceability of changes.

Enable diagnostic logging and Azure Monitor integration
Collect control plane and cluster logs through Azure Monitor and Log Analytics for centralised analysis. This provides the audit data required for compliance reporting and investigations.

Implement network policies for workload isolation
Use Kubernetes Network Policies or Azure Network Policy Manager to enforce segmentation between workloads. This ensures compliance with data isolation requirements and prevents unauthorised access.

Align platform configuration with regulatory requirements
Manage infrastructure and application changes through controlled workflows such as GitOps or CI/CD with approval gates. This ensures all changes are reviewed, traceable, and compliant with internal policies.

Implement configuration baseline validation and drift control
Continuously validate cluster configuration against approved baselines and detect drift from expected state. This ensures long-term compliance and prevents silent misconfigurations.

Define workload separation and tenancy model
Define how workloads are separated using namespaces, node pools, or clusters based on risk and compliance requirements. This ensures appropriate isolation between teams, applications, and environments.

Separate system and user node pools
Use dedicated system node pools with taints and tolerations for control plane components and avoid mixing with application workloads. This improves cluster stability and aligns with governance and operational best practices.

Enable and retain control plane audit logs
Send Kubernetes API server and control plane logs to Azure Monitor or a SIEM and retain them according to compliance requirements. This provides audit evidence and supports investigations and regulatory reporting.

Define and enforce tagging standards across resources
Apply consistent tagging for ownership, environment, cost allocation, and compliance tracking. This enables governance reporting and aligns resources with organisational structures.

Enable cluster encryption at rest and in transit
Configure encryption for etcd using Azure-managed or customer-managed keys and enforce TLS for all communications. This protects sensitive data and supports compliance with frameworks such as GDPR, HIPAA, and SOC 2.

Establish regular compliance audits and assessments
Perform periodic assessments using Microsoft Defender for Cloud, Azure Policy reports, and other tools. This ensures compliance is continuously validated and not treated as a one-time activity.

Bottom Line

Preparing AKS for production is not a one-time activity. As workloads evolve, configurations drift, and platform capabilities change, maintaining a reliable, secure, and efficient cluster requires continuous review and adjustment.

This checklist provides a structured way to assess your current state, identify gaps, and prioritise improvements across key areas of your platform. Used regularly, it helps ensure your AKS environment remains aligned with best practices and operational requirements.

AKS Assessment

Intercept runs a comprehensive scan of your AKS clusters, analysing configuration, security posture, and resource efficiency using 7 days of Prometheus metrics.
Get a detailed health report with prioritised recommendations to improve reliability, reduce costs, and align with Kubernetes best practices.

contact@intercept

Plan the assessment